Machine Learning Project: Pneumonia X-Ray images

Benedetta Pacilli
Computer Science and Engineering
Curriculum of Intelligent Embedded Systems
University of Bologna
Email: benedetta.pacilli@studio.unibo.it
ID: 0001136705

Valentina Pieri
Computer Science and Engineering
Curriculum of Intelligent Embedded Systems
University of Bologna
Email: valentina.pieri5@studio.unibo.it
ID: 0001137366

***Abstract*—In medical diagnostics, X-ray images are widely used to detect and diagnose diseases. In particular, chest X-ray images are used to detect pneumonia, a lung infection that different agents can cause. Pneumonia can be life-threatening, especially for people with a weak immune system, therefore it is important to detect pneumonia as soon as possible, to start the treatment and avoid complications. This Machine Learning project aims to build a robust classifier that can distinguish between normal and pneumonia X-ray thoracic images.**
The project focuses on studying and employing various machine learning and deep learning algorithms to create multiple classifiers capable of accurately distinguishing between a normal and a pneumonia-affected patient.

1. Introduction

MEDICAL diagnostics heavily rely on X-ray images for the detection and diagnosis of various diseases. Among these, chest X-ray images play a crucial role in identifying pneumonia, a potentially life-threatening lung infection caused by different agents. Timely detection of pneumonia is critical, especially for individuals with compromised immune systems, as it enables prompt treatment initiation and helps prevent complications.
This Machine Learning project is dedicated to constructing a robust classifier capable of distinguishing between normal and pneumonia-affected X-ray thoracic images. The project's primary focus involves a comprehensive study and application of various machine learning and deep learning algorithms. The objective is to develop multiple classifiers with the capability to accurately identify patients with pneumonia from normal cases.

THE project's code is organized into four main sections, each serving a distinct purpose:

1) **Data Import**: The initial phase involves importing the dataset, specifically the Chest X-ray images (Pneumonia) from *Kaggle*. The dataset comprises 5,863 X-ray images categorized into Pneumonia and Normal cases. We consider the Pneumonia class, which is later shown to be the *rare* class, as our *Positive* class. The data is divided into three folders: *train*, *test*, and *val*.
2) **Data Exploration**: This section delves into exploring and analyzing the dataset. Notable tasks include redistributing data for balance, visualizing images, and assessing the distribution of classes, sizes, and brightness within the dataset.
3) **Machine Learning**: The project progresses to training and testing various machine learning models. This section encompasses feature extraction, metrics function definition, and creation of different machine learning models (*SVM* [1], *Random Forest* [2], *Decision Tree*, *AdaBoost* [3], and *xgBoost* [4]), and evaluation of the best-performing models.
4) **Deep Learning**: The final section extends the exploration of deep learning models. This includes data preprocessing (with color *jittering* and layer augmentation [8]), building a convolutional neural network (*CNN* [5]) model architecture, training the model with preprocessed data, and running inference to assess model performance.

The significance of this project lies in its potential to contribute to the automation and accuracy of pneumonia diagnosis through advanced machine learning and deep learning techniques. The development of effective classifiers can aid healthcare professionals in swiftly identifying pneumonia cases from chest X-ray images, thereby facilitating timely intervention and reducing the associated risks.
The subsequent sections of this paper will delve into the specific details of each project phase.

Benedetta Pacilli, Valentina Pieri

January 26, 2024

2. Related Works

The leading inspiration for our work was **CheXNet** [9], a CNN capable of detecting pneumonia from chest X-rays. The researchers evaluated their work by comparing their algorithm's classification results with the classifications made by several practicing academic radiologists. The results obtained are sensational as they found that CheXNet exceeds average radiologist performance.
We thought it interesting to challenge ourselves with such a task, to see how well our *beginner-level* models could perform.
Due to several inherent differences between the two projects that could not be eliminated and would have significantly impacted the comparison, it was not possible to make a practical comparability study between our work and theirs:

- They used an image dataset containing over 100.000 elements, ours has approximately 6000 entries.
- For CheXNet, a team of professional radiologists was employed to annotate their test set while we used an already annotated dataset from Kaggle.
- Their architecture is a 121-layer CNN, whilst our Convolutional Neural Network is made of 10 layers.
- We focused on a binary classification to state if a patient either has pneumonia or not, without making differences between the diverse pneumonia types. On the other hand, CheXNet classification outputs a vector of binary labels indicating the absence or presence of different 14 pathologies.

3. Proposed Method

3.1. Data analysis

The initial phase involved the import and analysis of the dataset. We first realized an imbalance between the *train*, *test*, and *val* datasets. To address this issue and ensure a more fair distribution, we redistributed data as a $70/15/15$ split across the partitions.
The decision to redistribute data was mainly motivated by the concern over the limited size of the *val* dataset, as it has a crucial role in assessing model performance.
The subsequent step involved visualizing and analyzing illustrative examples of the datasets. A closer examination of the images revealed some anomalies, addressed in the preprocessing phase. For instance, as illustrated in Figure 1,

Figure 1. A x-ray image example of a healthy patient.

certain images exhibit a framing issue where non-essential parts of the body, such as the chin, are included alongside the chest. Another consideration concerns the alignment of

Figure 2. A x-ray image example of a patient affected by pneumonia.

subjects within the X-ray images. Comparing Figure 1 and Figure 2, it is notable how the former exhibits a subtle tilt to the right, while the latter manifests a slight inclination to the left.
Additionally, images present variations in brightness levels. Figures 3 and 4 illustrate instances where certain images exhibit pronounced darkness, while others display heightened luminosity.

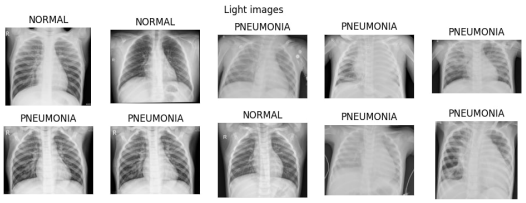

Figure 3. Light sample images

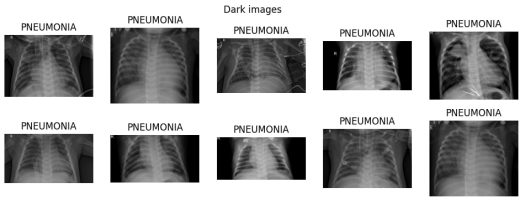

Figure 4. Dark sample images

The images portrayed in Figure 3 and Figure 4 differ not only in brightness but also in their dimensional attributes, encompassing both height and width. By selecting and plotting random images with particularly small dimensions we discovered several cropped pictures in each dataset, especially within the pneumonia class, as demonstrated in Figure 5. A plausible rationale is the potential utility of having zoomed-in representations of pneumonia scans. Finally, we

Figure 5. Example of cropped images found in the datasets.

checked the distribution of the two classes, *PNEUMONIA* and *NORMAL*, across the three datasets. The observation revealed a predominant presence of scans from healthy

patients in each dataset.
This class imbalance could lead to potential biases in the model, but no actual issues were encountered during the training and evaluation phases. The model's performance was not adversely affected, signaling robustness in handling the imbalanced dataset.

3.2. Machine Learning

After analyzing the data, we focused on the training and testing of diverse machine learning models.
We began with feature extraction to transform raw image data into a format that is favorable for model training. Features are essential characteristics or patterns within the data that facilitate the learning process for the machine learning models. Extracting features aids in reducing data dimensionality, allowing models to focus on the most informative aspects of the data and improving computational efficiency.
Following the laboratory lessons, we made three feature extraction methods available: **HOG**, **LBP**, and **pixel normalization**.
To evaluate our models we employed the following metrics: **Precision**, **Recall**, **F1-Score**, **Confusion Matrix**, and **Accuracy**.

We experimented with the following *Machine Learning* algorithms.

- **SVM (Support Vector Machine):** A powerful supervised machine learning algorithm for classification and regression tasks. It works by finding the optimal hyperplane that maximizes the margin between classes in a high-dimensional space.
- **Decision Tree:** A tree-like model of decisions, where an internal node represents a decision based on the value of a particular feature, each branch represents an outcome of the decision, and each leaf node represents a class label.
- **Random Forest:** An ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (classification) of the individual trees.
- **AdaBoost:** A boosting algorithm that combines multiple weak classifiers to create a strong classifier. As the base estimator, we chose a Decision Tree classifier. It assigns weights to data points and focuses on the mistakes made by the previous models to improve accuracy.
- **xgBoost:** An efficient and scalable implementation of gradient boosting. It is particularly popular for its speed and performance.

Here we report the choice of parameters made for each algorithm:
SVM

- `gamma`: It controls the influence of a single training example, it was set to 0.0001
- `C`: Regularization parameter that controls the trade-off between smooth decision boundary and classifying training points correctly, it was set to 100.
- `Kernel`: Non-linear **rbf** kernel was employed. **rbf** stands for Radial Basis Function kernel and it is a popular kernel suitable for non-linear data as it maps input data into high-dimensional space

Random Forest

- `n_jobs`: By setting this parameter to -1 we utilized all processors available
- `bootstrap`: Set to `True`. It refers to the technique of resampling with replacement from the original dataset to create multiple training datasets for building individual decision trees. This helps introduce randomness and diversity among the trees, which can improve the overall performance of the Random Forest by reducing overfitting and increasing robustness.
- `criterion`: Set to Entropy criterion to measure impurity for decision trees
- `max_features`: This parameter, in our case set to the square root of the number of features, determines the maximum number of features each tree is allowed to consider when making a split at a node. Restricting the features considered at each split encourages the trees in the ensemble to be more independent, leading to a more robust and generalizable model.
- `n_estimators`: The total number of decision trees, in our case 200.
- `max_depth`: The maximum depth of each decision tree in the forest, used to control model complexity. We set it to 100

Decision Tree:

- `criterion`: We chose the Entropy criterion here as well.
- `max_depth`: We set also this maximum depth at 100.

AdaBoost:

- `estimator`: This parameter in AdaBoost refers to the base estimator, which is the underlying weak learner used for building the ensemble. In our context, the base estimator chosen was the `DecisionTreeClassifier` with the `max_depth` parameter set to 1.
- `n_estimators`: the total number of weak learners in the ensemble was set to 50.

xgBoost:

- `objective`: was set to Multi:softmax learning objective. It specifies the learning objective or loss function for the training process.
- `num_class`: Number of classes that we have, 2 in this case.
- `max_depth`: Maximum tree depth set to 3

- `learning_rate`: Set to 0.00001. The learning rate controls the contribution of each boosting tree to the final prediction. A lower learning rate makes the training process more conservative by taking smaller steps toward minimizing the loss function at each iteration. This can help prevent overfitting and improve the generalization of the model. However, a smaller learning rate typically requires a higher number of boosting trees to achieve a good performance
- `n_estimators`: Total number of boosting trees set to 100
- `subsample`: This specifies the fraction of training data used for each boosting iteration, we set it to 0.8
- `colsample_bytree`: It represents the fraction of features used for each boosting iteration, 0.8 in our case.

3.3. Deep Learning

After experimenting with *Machine Learning* algorithms we also tested a *Deep Learning* model with differently preprocessed data.
We created two fundamental functions for the preprocessing phase:

- **preprocess_image_and_label**: It resizes images to 224×224, normalizes the pixels' values, and one-hot encodes the labels, to convert the categorical labels (*PNEUMONIA* and *NORMAL*) into a numerical format that can be fed into a neural network.
- **prepare_data**: It prepares the data, applying *color jittering* or *layers augmentation* based on the parameters. Color jittering randomly changes brightness, contrast, saturation, and hue to enhance model robustness to different lighting conditions. Layer augmentation randomly applies flips, rotations, and zooms to improve model generalization. The purpose of these types of data preprocessing is to resolve the issues discovered during the data analysis process.

During the *Deep Learning* phase we worked with tensors. Each tensor had four dimensions: batch size, height and width of the image, and the number of channels.

We worked with two groups of datasets: the first group contained *train*, *test*, and *val* datasets made of color-jittering augmented data; while the second one contained the datasets made of layers-augmented data.
We created four dedicated functions for the evaluation of the *Deep Learning* models:

- **plot_training_loss_accuracy_history**: Plots training loss and accuracy history, over the epochs.
- **evaluate_model**: Evaluates the model on the test data, calculating loss and accuracy.
- **calculate_metrics**: Calculates precision, recall, and F1-score.
- **plot_confusion_matrix**: Calculates and plots the confusion matrix.

The central part of the *Deep Learning* study is represented by the **build_model** function, which constructs the **CNN** model architecture using `tf.keras.Sequential` [6]. The architecture built employs the following layers:

- **Conv2D Layer:** A 2D convolutional layer applied to the input images. It performs a convolutional operation, capturing spatial patterns in the images. The ReLU (Rectified Linear Unit) activation function introduces non-linearity, enabling the model to learn complex relationships within the data.
- **MaxPooling2D Layer:** A 2D pooling layer that downsamples the input along its spatial dimensions. Using a 2x2 pooling size, as we did, reduces the spatial dimensions of the input by half, decreasing computational time and data complexity.
- **Conv2D + MaxPooling2D Combination:** Another set of 2D convolutional and pooling layers. This combination allows the model to extract more complex features from the images. Multiple combinations help the model learn hierarchical features. Although it may look like a redundancy, we initially tried to run training and validation with only one combination of Conv2D + MaxPooling2D layers. The model performed poorly so we decided to try with the two additional layers, resulting in a strongly improved performance.
- **Dropout Layer:** A regularization technique that randomly drops a specified percentage of neurons during training (in our case, 20%). Dropout prevents overfitting by forcing the network to rely on different pathways, enhancing generalization.
- **Flatten Layer:** Flattens the input, transforming it into a 1D array. This was necessary as the subsequent layer requires a 1D input.
- **Dense Layers:** We employed 4 fully connected (dense) layers in the neural network. The ReLU activation function is used for the first three dense layers to introduce non-linearity. The last dense layer uses the sigmoid activation function for binary classification (*NORMAL* or *PNEUMONIA*).

This function expects the number of neurons, in the first three dense layers, and the learning rate as parameters. We selected them by running a random search (`keras_tuner.tuners.RandomSearch` [7]) on both the color-jittering data and the layers-augmented data.
The hyperparameter tuning process, performed with multiple trials and epochs, led to the discovery of the best hyperparameters, based on *validation accuracy*.
The actual training and validation processes were performed on both datasets.

4. Results

In this section, we present the obtained results with each algorithm and strategy implemented.

4.1. Machine Learning

- **SVM**: overall, SVM was one of the best-performing models. The training accuracy was incredibly high (1.00) and, at the same time, we had a notably high accuracy on the *test* and *val* dataset as well, both at 0.96. Such a high accuracy value on the *train* set could signify that the model is overfitting. We tried different hyperparameter combinations but whenever we reached a lower accuracy for training, we ended up with a worse accuracy for the other datasets as well. Eventually, we decided to keep the combination that leads to 100% accuracy on training. With SVM, both precision and recall were around 97% while the confusion matrix presented an incredibly high number of **TP** and a satisfactory amount of **TN**.
- **Decision Tree**: Training accuracy was very high here as well while, *test* and *val* dataset accuracy were respectively 0.81 and 0.82. This behavior could be a signal of overfitting, different hyperparameter combinations were tried but with poor results. In the end, the Decision Tree-based model did not end up being one of our best models.
  Precision and recall were both around 87%; even if we don't consider it to be a *top-level* result, we believe it is still a satisfying one. The confusion matrix presented a good number of both **TP** and **TN**.
- **Random Forest**: Random Forest performed better than the Decision Tree algorithm, but not as good as SVM. We included it in the *best models' evaluation* part mainly to have an **ensemble method** that used **bagging**.
  With Random Forest, we obtained a precision of 0.89 and a recall result of 0.96. Even from the confusion matrix, we saw that the model performed quite well, especially on **TP**.
- **AdaBoost**: AdaBoost performed excellently on each dataset without showing alarming signs of overfitting. More specifically, we obtained 0.95, 0.90, and 0.90 respectively on training, testing, and validating data. For both precision and recall we got results around 0.93.
  Even though the AdaBoost confusion matrix was not as accurate as the SVM one, it was shown that the AdaBoost model performed better than the Random Forest model when identifying **TN**. On the other hand, the Random Forest model achieved slightly better results in terms of **TP**. This is another result that made us consider both AdaBoost and RandomForest as the best models, alongside the unbeaten SVM algorithm.
- **xgBoost**: xgBoost execution was good, but relatively poor if compared to the other algorithms. We obtained 84% of accuracy on each dataset and a precision and recall respectively of approximately 0.90 and 0.87.

Inference executions of **SVM**, **Random Forest**, and **AdaBoost** reflected the results obtained in the training and validation processes. All models showed robustness with the *test* images, data that they had not been exposed to during the training process.

4.2. Deep Learning

Both the training on data preprocessed with color jittering and the one on layers-augmented data presented slightly better accuracy results than any *Machine Learning* algorithm we tried.
For data preprocessed with color jittering, we obtained a training accuracy of approximately 94% and a validation accuracy of roughly 95%. By plotting the confusion matrix, we noticed how the model created, trained on the first kind of preprocessed data, performed better than SVM, RandomForest, and AdaBoost classifiers when recognizing *NORMAL* images; while it produced better results than DecisionTree and xgBoost classifiers both with *NORMAL* and Moving on to data preprocessed with layers augmentation, we obtained both a training and a validation accuracy of almost 94%.
Both model configurations performed very similarly: they both led to a precision and a recall of roughly 96% and even their confusion matrix were thoroughly very similar.

Inference was performed with both model configurations, resulting in satisfactory results. Both models, even if trained on different data, were able to correctly classify images from the *test* dataset with high confidence, showing robustness.

5. Conclusions

In conclusion, this machine and deep learning project aimed to develop robust classifiers for distinguishing between normal and pneumonia-affected X-ray thoracic images. Through extensive exploration of machine learning and deep learning algorithms, we successfully implemented various models and evaluated their performance on the provided dataset.
The machine learning phase involved the use of SVM, Decision Tree, Random Forest, AdaBoost, and xgBoost algorithms. Notably, SVM exhibited exceptional performance with high accuracy and robustness, followed closely by AdaBoost and Random Forest. These models showcased their effectiveness in identifying pneumonia cases from chest X-ray images.
In the deep learning phase, convolutional neural networks (CNNs) were employed to further enhance the classification task. The models, trained on both color-jittering and layers-augmented data, demonstrated impressive accuracy and generalization capabilities. The CNNs slightly outperformed traditional machine learning algorithms, providing promising results for automated pneumonia diagnosis.

5.1. Review of Group Work

The collaboration between us was fundamental to the success of this project. We both actively contributed to every aspect of this work: each decision, research, study, trial, and coding activity was made together with both members present.
For data acquisition and analysis we worked jointly while for the *Machine Learning* and *Deep Learning* parts we worked in parallel on different aspects.
Benedetta Pacilli handled the code regarding SVM and AdaBoost models creation and testing; while Valentina Pieri worked on Decision Tree, Random Forest, and xgBoost models creation and testing.
For the *Deep Learning section* we divided the workload based on the type of preprocessing. Benedetta Pacilli worked on data preprocessing with color jittering and the relative training and evaluation; while Valentina Pieri focused on data preprocessing with layers augmentation and the related training and evaluation phases. The deep learning base model architecture was conceived together and agreed upon. Regardless of the workload division, we always made joint decisions, engaged in discussions, and collectively addressed challenges. Our partnership was characterized by open communication, shared responsibility, and a mutual commitment to the project's success.

References

[1] Corinna Cortes and Vladimir Vapnik, *Support-Vector Networks*, *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995. https://doi.org/10.1007/BF00994018.

[2] Leo Breiman, *Random Forests*, *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001. https://doi.org/10.1023/A:1010933404324.

[3] Yoav Freund and Robert E. Schapire, *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*, *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997. https://doi.org/10.1006/jcss.1997.1504.

[4] Tianqi Chen and Carlos Guestrin, *XGBoost: A Scalable Tree Boosting System*, *CoRR*, vol. abs/1603.02754, 2016. http://arxiv.org/abs/1603.02754.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, *Advances in Neural Information Processing Systems*, vol. 25, 2012. https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[6] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek Gordon Murray, Benoit Steiner, Paul A. Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, Xiaoqiang Zhang, *TensorFlow: A system for large-scale machine learning*, *CoRR*, vol. abs/1605.08695, 2016. http://arxiv.org/abs/1605.08695.

[7] James Bergstra and Yoshua Bengio, *Random Search for Hyper-Parameter Optimization*, *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012. http://jmlr.org/papers/v13/bergstra12a.html.

[8] Connor Shorten and Taghi M. Khoshgoftaar, *A survey on Image Data Augmentation for Deep Learning*, *Journal of Big Data*, vol. 6, pp. 1–48, 2019. https://api.semanticscholar.org/CorpusID:195811894.

[9] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Yi Ding, Aarti Bagul, Curtis P. Langlotz, Katie S. Shpanskaya, Matthew P. Lungren, Andrew Y. Ng, *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*, *CoRR*, vol. abs/1711.05225, 2017. http://arxiv.org/abs/1711.05225.